PCS 2428 / PCS 2059
Inteligência Artificial

Prof. Dr. Jaime Simão Sichman
Prof. Dra. Anna Helena Reali Costa

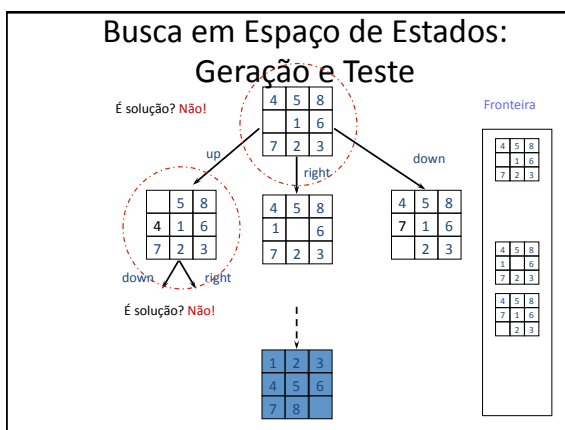
Busca Cega

Busca em Espaço de Estados: Geração e Teste

- *Fronteira* do espaço de estados
  - nós (estados) a serem expandidos no momento.
  - inicialmente, a fronteira contém o estado inicial do problema.

Algoritmo Genérico de Busca em Espaço de Estados:

1. Selecionar o primeiro nó (estado) da *fronteira* do espaço de estados;
   - se a fronteira está vazia, o algoritmo termina com falha.
2. Testar se o nó é um estado final (solução):
   - se "sim, então retornar nó - a busca termina com sucesso.
3. Gerar um novo conjunto de estados pela aplicação dos operadores ao estado selecionado;
4. Inserir os nós gerados na *fronteira*, de acordo com a estratégia de busca usada, e voltar para o passo (1).

Busca em Espaço de Estados: Geração e Teste

Busca em Espaço de Estados: Implementação

- Espaços de Estados
  - podem ser representados como uma árvore onde os estados são nós e as operações são arcos.
- Os nós da árvore podem guardar mais informação do que apenas o estado:

  → são uma estrutura de dados com 5 componentes:
  1. o estado correspondente
  2. o seu nó pai
  3. o operador aplicado para gerar o nó (a partir do pai)
  4. a profundidade do nó
  5. o custo do nó (desde a raiz)

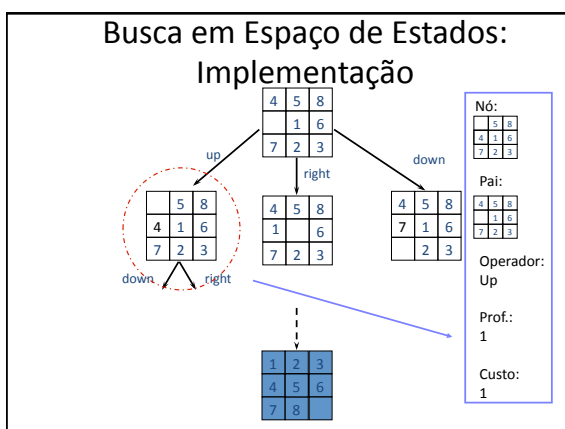
Busca em Espaço de Estados: Implementação

Nó:

Pai:

Operador: Up

Prof.: 1

Custo: 1

Busca em Espaço de Estados:Algoritmo

função **Busca-Genérica (*problema*, Função-Insere)**
  retorna **uma solução ou falha**

```
fronteira ← Faz-Fila (Faz-Nó (Estado-Inicial [problema] ) )
loop do
    se fronteira está vazia então retorna falha
    nó ← Remove-Primeiro (fronteira)
    se Teste-Término [problema] aplicado a Estado [nó] tiver sucesso
    então retorna nó
    fronteira ← Função-Insere (fronteira, Operadores [problema])
end
```

Função-Insere: controla a ordem de inserção de nós na fronteira do espaço de estados.

Métodos de Busca

- Busca não informada (busca cega / exaustiva)
  - Não tem informação sobre qual sucessor é mais promissor para atingir a meta.
  - Estratégias de Busca (ordem de expansão dos nós):
    - busca em largura
    - busca de custo uniforme
    - busca em profundidade
    - busca em profundidade limitada
    - busca em prof. com aprofundamento iterativo
    - busca bidirecional
- Busca heurística (busca informada)
  - Possui informação (estimativa) de qual sucessor é mais promissor para atingir a meta.

7

Busca não informada

- A busca não informada (ou busca cega) não possui estimativas sobre qual sucessor é mais promissor para atingir a meta.
- Fronteira : todos os nós gerados e ainda não expandidos (ou visitados) da árvore de busca.

8

Estratégias de Busca Cega

- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos
- Busca com Conhecimento Incompleto

9

Estratégias de Busca Cega: Exemplo

Problema: sair de A e chegar até Z ....

B E A D H I F C G J K

Busca em Largura

- Ordem de expansão dos nós:
  1. Nó raiz
  2. Todos os nós de profundidade 1
  3. Todos os nós de profundidade 2, etc...

Fronteira = FIFO (first-in-first-out)

➔ insere no fim da fila

11

Desempenho da busca em largura

- Completa?
  - Se b finito, é completa: se um nó-meta estiver a uma profundidade d, a busca em largura sempre irá encontrá-lo.
- Ótima?
  - Nem sempre – caminho mais curto (nó-meta mais próximo da raiz) ≠ melhor caminho
  - É ótima se o custo do caminho for uma função não-decrescente da profundidade do nó (ex: todas ações têm mesmo custo)

12

Desempenho da busca em largura

- Complexidade de tempo
  - Meta em d, cada nó tem b filhos. No pior caso, vem (teste ao expandir cada nó):

$$1 + b + b^2 + b^3 + ... + b^d + (b^{d+1} - b) = \mathcal{O}(b^{d+1})$$

- Complexidade de espaço
  - Mantém todos nós gerados (ou está na fronteira ou é ancestral e está na lista de visitados)

$$1 + b + b^2 + b^3 + ... + b^d + (b^{d+1} - b) = \mathcal{O}(b^{d+1})$$

13

Desempenho da Busca em Largura

- Esta estratégia só dá bons resultados quando a *profundidade* da árvore de busca é *pequena*

Exemplo:

fator de expansão b = 10

1.000 nós gerados por segundo

cada nó ocupa 100 bytes

Profundidade	Nós	Tempo	Memória
0	1	1 milissegundo	100 bytes
2	111	0.1 segundo	11 quilobytes
4	11111	11 segundos	1 megabytes
6	10^6	18 minutos	111 megabytes
8	10^8	31 horas	11 gigabytes
10	10^{10}	128 dias	1 terabyte
12	10^{12}	35 anos	111 terabytes
14	10^{14}	3500 anos	11111 terabytes

Busca de Custo Uniforme

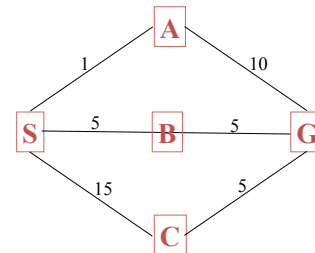
- Modifica a busca em largura:
  - Em vez de expandir o nó gerado primeiro, expande o nó da fronteira com menor custo de caminho (da raiz ao nó)

  Fronteira ➔ insere em ordem crescente de custo

- Não se importa com o número de passos, mas com o custo total
- g(n) dá o custo do caminho da raiz ao nó n
  - Na busca em largura: ***g(n) = profundidade (n)***

15

Busca de Custo Uniforme

- Exercício: gerar a árvore de busca usando busca de custo uniforme. Suponha estado inicial S e estado final G.

16

Busca de Custo Uniforme
Fronteira do exemplo anterior

- F = {S}
  - testa se S é o estado objetivo, expande-o e guarda seus filhos A, B e C ordenadamente (segundo custo) na fronteira
- F = {A, B, C}
  - testa A, expande-o e guarda seu filho G_A ordenadamente
  - **obs.:** o algoritmo de geração guarda na fronteira todos os nós gerados, testando se um nó é o objetivo apenas quando ele é retirado da lista (i.e., visitado ou expandido)!

17

Busca de Custo Uniforme
Fronteira do exemplo anterior

- F= {B, G_A, C}
  - testa B, expande-o e guarda seu filho G_B ordenadamente
- F= {G_B, G_A, C}
  - testa G_B e pára!

18

Busca de Custo Uniforme

Desempenho da Busca de Custo Uniforme

- Completa? Só *se custo de cada ação ≥ ε , ∀ n*
  - *ε é uma constante pequena positiva*
    - *Loop* infinito: quando expande nó que tem ação de custo=0 levando de volta ao mesmo nó.
- Ótima? *Só se g(sucessor(n)) > g(n)*
  - *custo **no mesmo caminho** sempre cresce (i.e., não tem ação com **custo negativo ou 0**)*
- Complexidade de tempo e de espaço
  - *C*=custo da solução ótima (custo de cada ação≥ε)*

➔ Pior caso: $\mathcal{O}(b^{1+\lfloor C^*/\varepsilon \rfloor})$, o que pode ser bem maior que b^d

Quando todos os custos são iguais, $b^{1+\lfloor C^*/\varepsilon \rfloor} = b^d$

Busca em Profundidade

- Ordem de expansão dos nós:
  1. Nó raiz
  2. Primeiro nó de profundidade 1
  3. Primeiro nó de profundidade 2, etc…

Fronteira = LIFO (last-in-first-out)

➔ insere no início da fila

Busca em Profundidade

F=[IEC] F=[EC] …

Neste exemplo, nós com profundidade 3 não têm sucessores.
Nós sem sucessores e já expandidos são "apagados".

Desempenho da Busca em Profundidade

- Esta estratégia **não é *completa*** (caminho pode ser infinito) **nem é *ótima***.Se usar uma estratégia que não permite estados repetidos nem caminhos redundantes, ela é completa.
- Complexidade espacial:
  - mantém na memória o caminho que está sendo expandido no momento, e os nós irmãos dos nós no caminho para possibilitar o retrocesso (*backtracking*)
  - Apaga subárvores já visitadas
  - Para espaço de estados com fator de ramificação b e **profundidade máxima m** (m pode ser >> d), requer **bm+1** de memória ➔ $\mathcal{O}$ (bm)

Desempenho da Busca em Profundidade

- Complexidade temporal: $\mathcal{O}(b^m)$, no pior caso.
  - Para problemas com várias soluções, esta estratégia pode ser bem mais rápida do que a busca em largura.
  - Esta estratégia deve ser evitada quando as árvores geradas são muito *profundas* ou geram *caminhos infinitos*.

Variante: **Busca com Retrocesso (backtracking search)**

- Parecida com BP, mas **somente UM sucessor é gerado em cada iteração**
  - na BP, todos os sucessores são gerados na expansão do nó pai
- Portanto, requer só $\mathcal{O}$(m) de memória
  - BP requer $\mathcal{O}$(bm) de memória
- Restrição: deve ser capaz de retornar ao pai e criar o novo sucessor

25

Busca com Aprofundamento Limitado

- Evita o problema de árvores não limitadas ao impor um limite máximo (l) de profundidade para os caminhos gerados por uma BP.
  - O domínio do problema estabelece a profundidade limite.
  - Problema: definir limite l adequado!
- Completa? Somente se l ≥ d.
- Ótima? Não, exceto se l = d.
- Complexidade espacial: $\mathcal{O}$ (b . l)
- Complexidade temporal: $\mathcal{O}$ (b^l) no pior caso.

BP é caso particular, com l = ∞

26

Busca com Aprofundamento Iterativo (BAI)

- Tenta limites de BP com valores crescentes, partindo de zero, até encontrar a primeira solução (em d).
  - Combina vantagens da busca em largura (BL) com as da busca em profundidade (BP).
  - Em geral, é a estratégia preferida de busca cega para quando o espaço de estados é muito grande e a profundidade da solução d é desconhecida.
- Possui uma variante, a **Busca com Comprimento Iterativo (BCI)** – analogia entre BAI e BL, e BCI e Custo Uniforme: usa incremento iterativo do **custo** do caminho em vez de incremento na profundidade (mas BCI não é eficiente!)

27

Desempenho da BAI

- Completa? Sim se b for finito (idem BL).
- Ótima? Sim se o custo do caminho for uma função crescente com a profundidade do nó (idem BL).
- Complexidade espacial:

  $\mathcal{O}$ (bd) (idem BP)

28

Desempenho da BAI

- Complexidade temporal: nós na profundidade da menor solução (d) são gerados 1 vez, em d-1 são gerados 2 vezes, ... na profundidade 1 são gerados d vezes:

$$(d)b + (d-1)b^2 + (d-2)b^3 + ... + (1)b^d = \mathcal{O}(b^d)$$

OBS: BL gera alguns nós em d+1 e BAI não gera.

BL: $\mathcal{O}$ (b^{d+1})

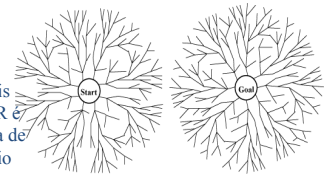
Na realidade, BAI é mais rápida que BL.

29

Busca Bidirecional (1)

- Busca em duas direções (duas buscas simultâneas):
  - para frente, a partir do nó inicial, e
  - para trás, a partir do nó final (objetivo)
- A busca pára quando o nó a ser expandido por uma busca se encontra na fronteira da outra busca.
- **Motivação:**

$b^{d/2} + b^{d/2} < b^d$.

Ou: a área de dois círculos de raio R é menor que a área de um círculo de raio 2R

Busca Bidirecional (2)

- Para BL nas duas direções: $\mathcal{O}(b^{d/2})$ no tempo e no espaço. Completude e otimalidade: idem BL.
  - É possível utilizar *estratégias* diferentes em cada direção da busca (podendo sacrificar desempenho)
- Porém, encadeamento reverso (da meta para o início) só é possível se todas as ações no espaço de estado forem reversíveis.
- Outro problema: quando há vários estados-meta ou quando é muito difícil computar os estados-meta pelo teste de término (ex. estados para cheque-mate).

31

Evitando Estados Repetidos (1)

- Problema geral em Busca
  - expandir estados já previamente encontrados e expandidos
- É inevitável quando há operadores reversíveis
  - ex. encontrar rotas, canibais e missionários, 8-números, etc.
  - a árvore de busca é potencialmente infinita

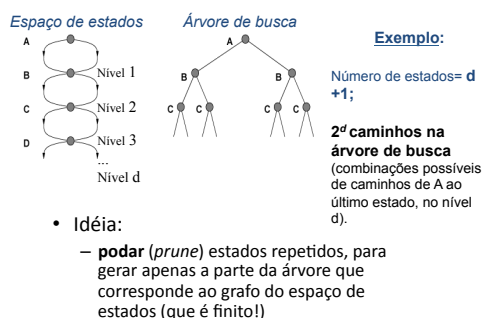
32

Evitando Estados Repetidos (2)

Exemplo:

Número de estados= **d +1;**

2^d caminhos na árvore de busca (combinações possíveis de caminhos de A ao último estado, no nível d).

- Idéia:
  - **podar** (*prune*) estados repetidos, para gerar apenas a parte da árvore que corresponde ao grafo do espaço de estados (que é finito!)

33

Evitando Estados Repetidos (3)

- **Problema**: deve armazenar <u>todos</u> nós gerados!
  - Além da lista de fronteira (também chamada de ***open list***), os algoritmos precisam da lista de nós visitados / expandidos (***closed list***)
  - Cada nó gerado é comparado com aqueles da *closed list*: se for repetido, descarta aquele de caminho com custo pior.
  - Pode ser implementado mais eficientemente com *hash tables*
  - BP e BAI: perdem propriedade de complexidade linear no espaço.

34

Busca com Conhecimento Incompleto

- E se o ambiente <u>não</u> for totalmente observável, determinístico e o agente não souber as conseqüências de suas ações?

1. Problemas **<u>conformantes</u>** (sem sensores): não sabe seu estado inicial e, assim, cada ação poderia levar a muitos estados sucessores.
2. Problemas **<u>contingenciais</u>**: quando o ambiente é parcialmente observável ou suas ações possuem incertezas. Se a incerteza é causada por ações de outro agente, é um problema com adversário.
3. Problemas de **<u>exploração</u>**: quando os estados e a dinâmica do ambiente são desconhecidos, o agente precisa atuar para descobrí-los (caso extremo dos problemas contingenciais).

35

Problemas Conformantes (sem sensores)

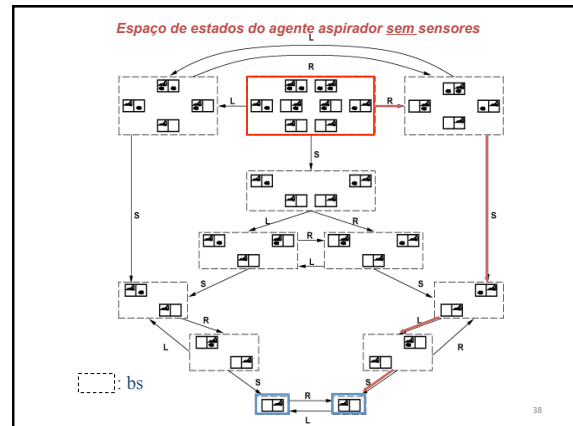
- Estado de crença (*belief state – bs*): **conjunto de estados em que o agente acredita estar.**
- **Busca ocorre no espaço de *bs:***
  - ações são aplicadas nos bs, gerando bs sucessores (∪ sucessores da ação aplicada a cada estado∈bs)
  - *<u>Solução</u>*: caminho que leva a um bs que <u>só</u> contenha estados-meta
  - Mesmo procedimento para ações não determinísticas

36

Exemplo com o agente aspirador

- Bs inicial =
{1, 2, 3, 4, 5, 6, 7, 8}
- Se aplicar a=Right em bs, vem:
bs' ={2, 4, 6, 8} ...
- Assim,
seq=[Right, Suck, Left, Suck]
é uma solução!

37

38

Problemas Contingenciais (1)

incertezas e observabilidade parcial

- Nenhuma seqüência fixa de ações garante a solução.
- Muitos problemas reais são contingenciais pois a predição exata é impossível.
- Planejamento condicional:
  - no meio da solução são inseridas ações de sensoriamento que direcionam a execução.
  - A solução normalmente é uma árvore, onde ações são selecionadas em função das contingências sensoriadas.
- Uso de abordagem probabilística

39

Problemas Contingenciais (2)

- Planejamento contínuo:
  - monitora e atualiza seu modelo do mundo continuamente, mesmo quando em deliberação;
  - assim que tiver um plano parcial, executa; revê metas, inclui novas metas, descarta metas, etc.
  - Projetado para interagir indefinidamente com o ambiente. Também usado em problemas de exploração.

40

Problemas Contingenciais (3)

- Monitoramento da execução e replanejamento:
  - agente planeja uma solução e a executa, monitorando a execução;
  - se ocorrer contingências e o plano precisar ser revisado, o agente replaneja a partir do estado que estiver.

contingência

Planejamento

Replanejamento

41